



COMP 346 - Winter 2022 Programming Assignment 3

Due Date: 11:59 PM - April 11, 2022

Warning: Redistribution or publication of this document or its text, by any means, is strictly prohibited. Additionally, publishing the solution publicly, at any point of time, will result in an immediate filing of an academic misconduct.

Dining Philosophers Problem

1 Objective

Implement the dining philosopher's problem using a monitor for synchronization.

2 Source Code

There are five files that come with the assignment. A soft copy of the code is available to download from the course web site. This time the source code is barely implemented (though compiles and runs). You are to complete its implementation.

2.1 File Checklist

Files distributed with the assignment requirements:

- common/BaseThread.java - unchanged
- DiningPhilosophers.java - the main()
- Philosopher.java - extends from BaseThread
- Monitor.java - the monitor for the system
- Makefile - take a look

3 Background

This assignment is a slight extension of the classical problem of synchronization – the Dining Philosophers problem. You are going to solve it using the Monitor synchronization construct built on top of Java's synchronization primitives. The extension refers to the fact that sometimes philosophers would like to talk, but only one (any) philosopher can be talking at a time while they are not eating. If you need help, consult the references at the bottom.

4 Tasks

Make sure you put comments for every task that involves coding to the changes that you've made. This will be considered in the grading process.

Task 1: The Philosopher Class

Complete the implementation of the Philosopher class, that is all its methods according to the comments in the code. Specifically, eat(), think(), talk(), and run() methods have to be implemented entirely.

Non-mandatory hints are provided within the code.

Task 2: The Monitor

Implement the Monitor class for the problem. Make sure it is correct, deadlock- and starvation-free implementation that uses Java's synchronization primitives, such as wait() and notifyAll(); no use

Redistribution or publication of this document or its text, by any means, is strictly prohibited.

of Semaphore objects is allowed. Implement the four methods of the Monitor class; specifically, `pickUp()`, `putDown()`, `requestTalk()`, and `endTalk()`. Add as many member variables and methods to monitor the conditions outlined below as needed:

1. A philosopher is allowed to pickup the chopsticks if they are both available. That implies having states of each philosopher as presented in your book. You might want to consider the order in which to pick the chopsticks up.

2. If a given philosopher has decided to make a statement, they can only do so if no one else is talking at the moment. The philosopher wishing to make the statement has to wait in that case.

Task 3: Variable Number of Philosophers

Make the application to accept a positive integer number from the command line, and spawn exactly that number of philosophers instead of the default one. If there are no command line arguments, the given default should be used. If the argument is not a positive integer, report this fact to the user, print the usage information as in the example below:

```
% java DiningPhilosophers -7.a
"-7.a" is not a positive decimal integer
```

```
Usage: java DiningPhilosophers [NUMBER_OF_PHILOSOPHERS]
%
```

Use `Integer.parseInt()` method to extract an int value from a character string.

Test your implementation with the varied number of philosophers. Submit your output from “make regression”.

5 Deliverables

IMPORTANT: You are allowed to work on a team of 2 students at most (including yourself!). Any teams of 3 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted for both members. You must make sure that you upload the assignment to the correct directory of **Programming Assignment 3** on Moodle. Assignments uploaded to the wrong directory will be discarded and no resubmission will be allowed.

- **Submission.**

- Create one zip file, containing the necessary files (.java, .txt and test cases). If the assignment is done individually, your file should be called `pa3_studentID`, where `pa3` is the number of the assignment and `studentID` is your student ID number. If the work is done in a team of 2 people, the zip file should be called `pa2_studentID1_studentID2` where `studentID1` and `studentID2` are the student ID numbers of each student.
- Upload your zip file on Moodle as *Programming Assignment 2* before midnight on the due date.

Redistribution or publication of this document or its text, by any means, is strictly prohibited.

- **IMPORTANT (Please read very carefully):**

A demo will take place with the markers afterwards. Markers will inform you about the details of demo time and how to book a time slot for your demo. If working in a group, both members must be present during demo time. Different marks may be assigned to teammates based on this demo.

- If you fail to demo, a zero mark is assigned regardless of your submission.
- If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.
- Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.

6 Grading Scheme

Grading Scheme:

=====		
T#	MX	MK

1	/2	
2	/6	
3	/2	

Total: /10

(T# - task number, MX - max (out of), MK - your mark)

7 References

1. Java API: <http://java.sun.com/j2se/1.3/docs/api/>
2. <http://java.sun.com/docs/books/tutorial/essential/TOC.html#threads>